林轩田《机器学习技法》课程笔记8 -- Adaptive Boosting

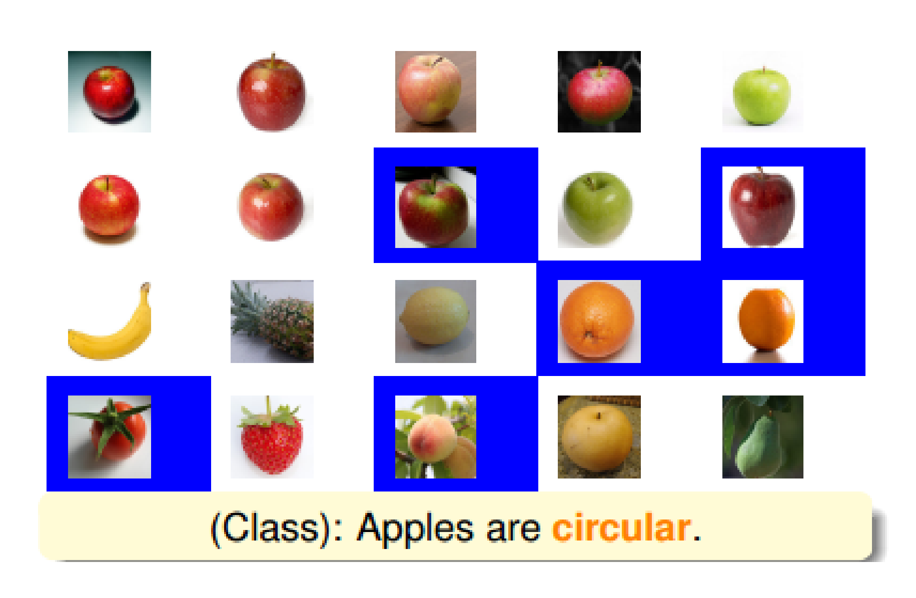
作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要开始介绍Aggregation Models，目的是将不同的hypothesis得到的g_t集合起来，利用集体智慧得到更好的预测模型G。首先我们介绍了Blending，blending是将已存在的所有g_t结合起来，可以是uniformly，linearly，或者non-linearly组合形式。然后，我们讨论了在没有那么多g_t的情况下，使用bootstrap方式，从已有数据集中得到新的类似的数据集，从而得到不同的g_t。这种做法称为bagging。本节课将继续从这些概念出发，介绍一种新的演算法。

Motivation of Boosting

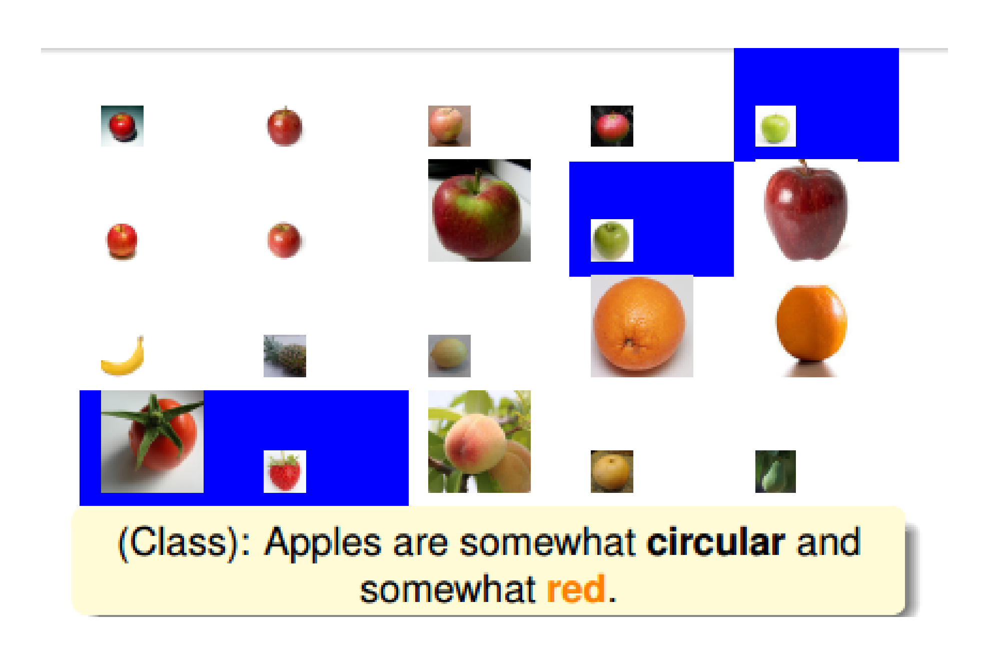
我们先来看一个简单的识别苹果的例子，老师展示20张图片，让6岁孩子们通过观察，判断其中哪些图片的内容是苹果。从判断的过程中推导如何解决二元分类问题的方法。

显然这是一个监督式学习，20张图片包括它的标签都是已知的。首先，学生Michael回答说：所有的苹果应该是圆形的。根据Michael的判断，对应到20张图片中去，大部分苹果能被识别出来，但也有错误。其中错误包括有的苹果不是圆形，而且圆形的水果也不一定是苹果。如下图所示：



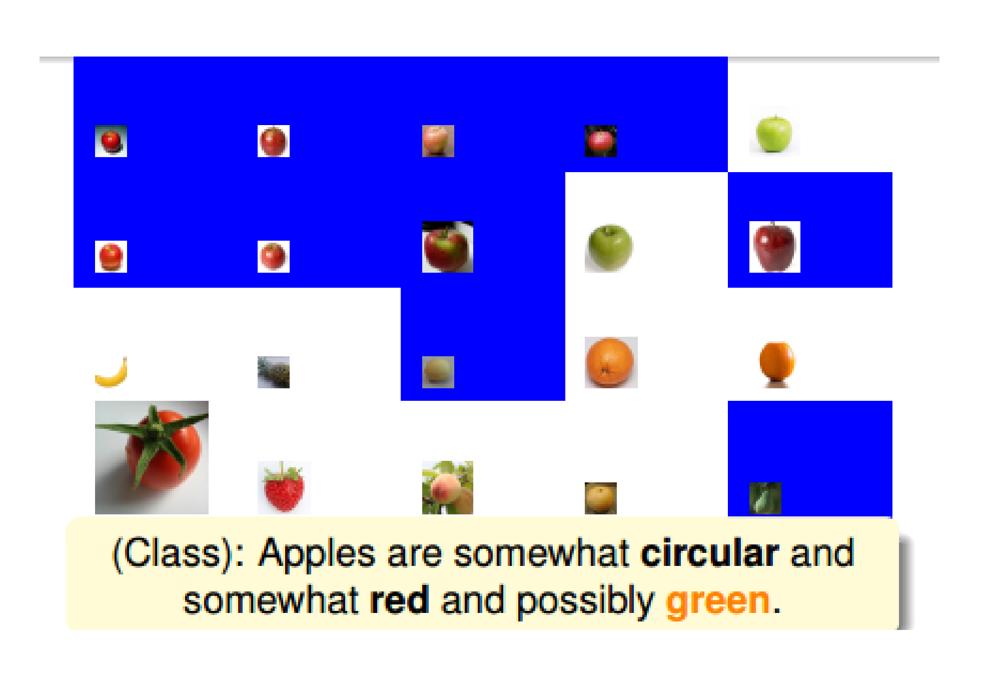
上图中蓝色区域的图片代表分类错误。显然，只用“苹果是圆形的”这一个条件不能保证分类效果很好。我们把蓝色区域（分类错误的图片）放大，分类正确的图片缩小，这样在接下来的分类中就会更加注重这些错误样本。

然后，学生Tina观察被放大的错误样本和上一轮被缩小的正确样本，回答说：苹果应该是红色的。根据Tina的判断，得到的结果如下图所示：



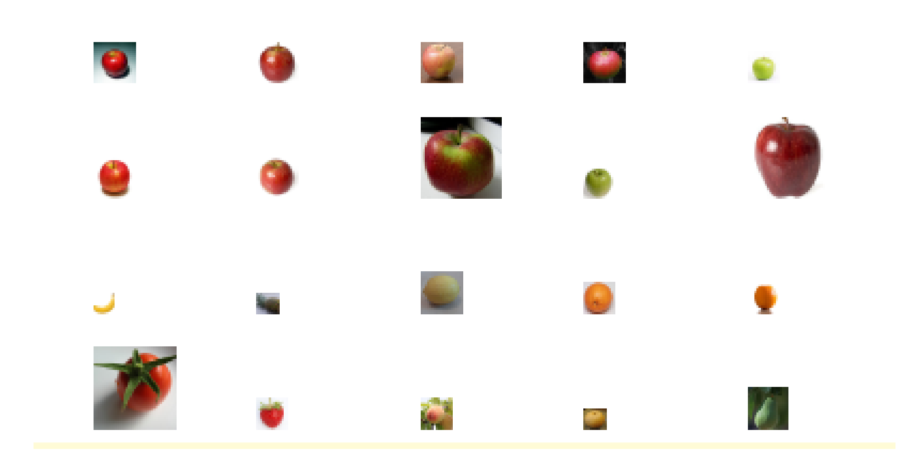
上图中蓝色区域的图片一样代表分类错误，即根据这个苹果是红色的条件，使得青苹果和草莓、西红柿都出现了判断错误。那么结果就是把这些分类错误的样本放大化，其它正确的样本缩小化。同样，这样在接下来的分类中就会更加注重这些错误样本。

接着，学生Joey经过观察又说：苹果也可能是绿色的。根据Joey的判断，得到的结果如下图所示：

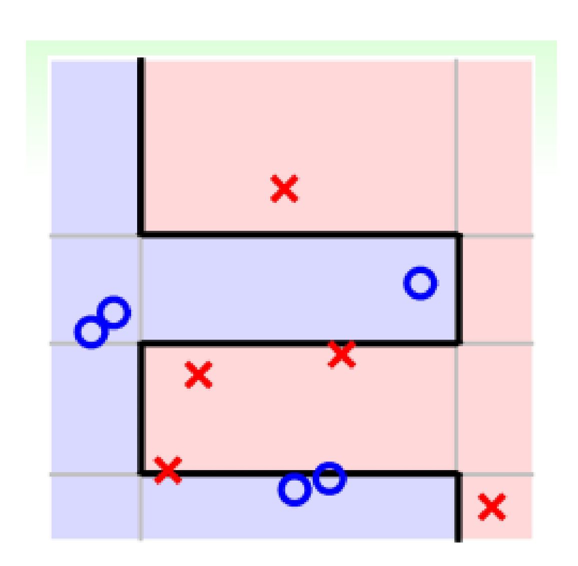


上图中蓝色区域的图片一样代表分类错误，根据苹果是绿色的条件，使得图中蓝色区域都出现了判断错误。同样把这些分类错误的样本放大化，其它正确的样本缩小化，在下一轮判断继续对其修正。

后来，学生Jessica又发现：上面有梗的才是苹果。得到如下结果：



经过这几个同学的推论，苹果被定义为：圆的，红色的，也可能是绿色的，上面有梗。从一个一个的推导过程中，我们似乎得到一个较为准确的苹果的定义。虽然可能不是非常准确，但是要比单一的条件要好得多。也就是说把所有学生对苹果的定义融合起来，最终得到一个比较好的对苹果的总体定义。这种做法就是我们本节课将要讨论的演算法。这些学生代表的就是简单的hypotheses g_t，将所有g_t融合，得到很好的预测模型G。例如，二维平面上简单的hypotheses（水平线和垂直线），这些简单g_t最终组成的较复杂的分类线能够较好地将正负样本完全分开，即得到了好的预测模型。



所以，上个苹果的例子中，不同的学生代表不同的hypotheses g_t；最终得到的苹果总体定义就代表hypothesis G；而老师就代表演算法A，指导学生的注意力集中到关键的

例子中（错误样本），从而得到更好的苹果定义。其中的数学原理，我们下一部分详细介绍。

- students: simple hypotheses g_t (like vertical/horizontal lines)
- (Class): sophisticated hypothesis G (like black curve)
- Teacher: a tactic learning algorithm that **directs the students to focus on key examples**

Diversity by Re-weighting

在介绍这个演算法之前，我们先来讲一下上节课就介绍过的bagging。Bagging的核心是bootstrapping，通过对原始数据集D不断进行bootstrap的抽样动作，得到与D类似的数据集$\hat{D}_t$，每组$\hat{D}_t$都能得到相应的g_t，从而进行aggregation的操作。现在，假如包含四个样本的D经过bootstrap，得到新的$\hat{D}_t$如下：

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}$$

$$\xrightarrow{\text{bootstrap}} \quad \tilde{\mathcal{D}}_t = \{(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_4, y_4)\}$$

那么，对于新的$\hat{D}_t$，把它交给base algorithm，找出E_{in}最小时对应的g_t，如下图右边所示。

$$E_{in}^{0/1}(h) = \frac{1}{4}\sum_{n=1}^{4}[y \neq h(x)]$$

由于$\hat{D}_t$完全是D经过bootstrap得到的，其中样本(x_1, y_1)出现2次，(x_2, y_2)出现1次，(x_3, y_3)出现0次，(x_4, y_4)出现1次。引入一个参数u_i来表示原D中第i个样本在$\hat{D}_t$中出现的次数，如下图左边所示。

$$E_{in}^{u}(h) = \frac{1}{4}\sum_{n=1}^{4}u_n^{(t)} \cdot [y_n \neq h(x)]$$

weighted E_{in} on $\mathcal{D}$

$$E_{\text{in}}^{\mathbf{u}}(h) = \frac{1}{4}\sum_{n=1}^{4} u_n^{(t)} \cdot [\![y_n \neq h(\mathbf{x}_n)]\!]$$

$$(\mathbf{x}_1, y_1),\ u_1 = 2$$
$$(\mathbf{x}_2, y_2),\ u_2 = 1$$
$$(\mathbf{x}_3, y_3),\ u_3 = 0$$
$$(\mathbf{x}_4, y_4),\ u_4 = 1$$

E_{in} on $\tilde{\mathcal{D}}_t$

$$E_{\text{in}}^{0/1}(h) = \frac{1}{4}\sum_{(\mathbf{x},y)\in\tilde{\mathcal{D}}_t} [\![y \neq h(\mathbf{x})]\!]$$

$$(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1)$$
$$(\mathbf{x}_2, y_2)$$
$$(\mathbf{x}_4, y_4)$$

参数u相当于是权重因子，当$\hat{D}_t$中第i个样本出现的次数越多的时候，那么对应的u_i越大，表示在error function中对该样本的惩罚越多。所以，从另外一个角度来看bagging，它其实就是通过bootstrap的方式，来得到这些u_i值，作为犯错样本的权重因子，再用base algorithn最小化包含u_i的error function，得到不同的g_t。这个error function被称为bootstrap-weighted error。

这种算法叫做Weightd Base Algorithm，目的就是最小化bootstrap-weighted error。

minimize (regularized)

$$E_{\text{in}}^{\mathbf{u}}(h) = \frac{1}{N}\sum_{n=1}^{N} u_n \cdot \text{err}(y_n, h(\mathbf{x}_n))$$

其实，这种weightd base algorithm我们之前就介绍过类似的算法形式。例如在soft-margin SVM中，我们引入允许犯错的项，同样可以将每个点的error乘以权重因子u_n。加上该项前的参数C，经过QP，最终得到$0 \le \alpha_n \le Cu_n$，有别于之前介绍的$0 \le \alpha_n \le C$。这里的u_n相当于每个犯错的样本的惩罚因子，并会反映到α_n的范围限定上。

同样在logistic regression中，同样可以对每个犯错误的样本乘以相应的u_n，作为惩罚因子。u_n表示该错误点出现的次数，u_n越大，则对应的惩罚因子越大，则在最小化error时就应该更加重视这些点。

SVM

$E_{in}^{\mathbf{u}} \propto C \sum_{n=1}^{N} u_n \widehat{\text{err}}_{\text{SVM}}$ by dual QP

$\Leftrightarrow$ adjusted upper bound $0 \leq \alpha_n \leq C u_n$

logistic regression

$E_{in}^{\mathbf{u}} \propto \sum_{n=1}^{N} u_n \text{err}_{\text{CE}}$ by SGD

$\Leftrightarrow$ sample $(\mathbf{x}_n, y_n)$ with probability proportional to u_n

其实这种example-weighted learning，我们在机器学习基石课程第8次笔记中就介绍过class-weighted的思想。二者道理是相通的。

知道了u的概念后，我们知道不同的u组合经过base algorithm得到不同的g_t。那么如何选取u，使得到的g_t之间有很大的不同呢？之所以要让所有的g_t差别很大，是因为上节课aggregation中，我们介绍过g_t越不一样，其aggregation的效果越好，即每个人的意见越不相同，越能运用集体的智慧，得到好的预测模型。

为了得到不同的g_t，我们先来看看g_t和g_{t+1}是怎么得到的：

$$g_t \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \left(\sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq h(\mathbf{x}_n)]\!] \right)$$

$$g_{t+1} \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \left(\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq h(\mathbf{x}_n)]\!] \right)$$

if g_t 'not good' for $\mathbf{u}^{(t+1)}$ $\Longrightarrow$ g_t-like hypotheses not returned as g_{t+1}

$\Longrightarrow$ g_{t+1} diverse from g_t

如上所示，g_t是由u_n^t得到的，g_{t+1}是由$u_n^{(t+1)}$得到的。如果g_t这个模型在使用$u_n^{(t+1)}$的时候得到的error很大，即预测效果非常不好，那就表示由$u_n^{(t+1)}$计算的g_{t+1}会与g_t有很大不同。而g_{t+1}与g_t差异性大正是我们希望看到的。

怎么做呢？方法是利用g_t在使用$u_n^{(t+1)}$的时候表现很差的条件，越差越好。如果在g_t作用下，$u_n^{(t+1)}$中的表现（即error）近似为0.5的时候，表明g_t对$u_n^{(t+1)}$的预测分类没有什么作用，就像抛硬币一样，是随机选择的。这样的做法就能最大限度地保证g_{t+1}会与g_t有较大的差异性。其数学表达式如下所示：

$$\text{idea: construct } \mathbf{u}^{(t+1)} \text{ to make } g_t \text{ random-like}$$

$$\frac{\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t+1)}} = \frac{1}{2}$$

乍看上面这个式子，似乎不好求解。但是，我们对它做一些等价处理，其中分式中分子可以看成g_t作用下犯错误的点，而分母可以看成犯错的点和没有犯错误的点的集合，即所有样本点。其中犯错误的点和没有犯错误的点分别用橘色方块和绿色圆圈表示：

$$\text{want: } \frac{\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t+1)}} = \frac{\blacksquare_{t+1}}{\blacksquare_{t+1} + \bullet_{t+1}} = \frac{1}{2}, \text{ where}$$

$$\blacksquare_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!], \bullet_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} [\![y_n = g_t(\mathbf{x}_n)]\!]$$

要让分式等于0.5，显然只要将犯错误的点和没有犯错误的点的数量调成一样就可以了。也就是说，在g_t作用下，让犯错的$u_n^{(t+1)}$数量和没有犯错的$u_n^{(t+1)}$数量一致就行（包含权重u_n^{t+1}）。一种简单的方法就是利用放大和缩小的思想（本节课开始引入识别苹果的例子中提到的放大图片和缩小图片就是这个目的），将犯错误的u_n^t和没有犯错误的u_n^t做相应的乘积操作，使得二者值变成相等。例如u_n^t of incorrect为1126，u_n^t of correct为6211，要让$u_n^{(t+1)}$中错误比例正好是0.5，可以这样做，对于incorrect $u_n^{(t+1)}$：

$$u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 6211$$

对于correct $u_n^{(t+1)}$：

$$u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 1126$$

或者利用犯错的比例来做，令weighted incorrect rate和weighted correct rate分别设为$\frac{1126}{7337}$和$\frac{6211}{7337}$。一般求解方式是令犯错率为ϵ_t，在计算$u_n^{(t+1)}$的时候，u_n^t分别乘以$(1-\epsilon_t)$和ϵ_t。

- need: $\underbrace{(\text{total } u_n^{(t+1)} \text{ of incorrect})}_{■_{t+1}} = \underbrace{(\text{total } u_n^{(t+1)} \text{ of correct})}_{●_{t+1}}$
- one possibility by **re-scaling (multiplying) weights**, if

(total $u_n^{(t)}$ of incorrect) = 1126 ; (weighted incorrect rate) = $\frac{1126}{7337}$	(total $u_n^{(t)}$ of correct) = 6211 ; (weighted correct rate) = $\frac{6211}{7337}$
incorrect: $u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 6211$	correct: $u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 1126$

'optimal' re-weighting under weighted incorrect rate ϵ_t:
multiply incorrect $\propto (1-\epsilon_t)$; multiply correct $\propto \epsilon_t$

Adaptive Boosting Algorithm

上一部分，我们介绍了在计算$u_n^{(t+1)}$的时候，u_n^t分别乘以$(1-\epsilon_t)$和ϵ_t。下面将构造一个新的尺度因子：

$$\diamond t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

那么引入这个新的尺度因子之后，对于错误的u_n^t，将它乘以$\diamond t$；对于正确的u_n^t，将它除以$\diamond t$。这种操作跟之前介绍的分别乘以$(1-\epsilon_t)$和ϵ_t的效果是一样的。之所以引入$\diamond t$是因为它告诉我们更多的物理意义。因为如果$\epsilon_t \leq \frac{1}{2}$，得到$\diamond t \geq 1$，那么接下来错误的$u_n^t$与$\diamond t$的乘积就相当于把错误点放大了，而正确的$u_n^t$与$\diamond t$的相除就相当于把正确点缩小了。这种scale up incorrect和scale down correct的做法与本节课开始介绍的学生识别苹果的例子中放大错误的图片和缩小正确的图片是一个原理，让学生能够将注意力更多地放在犯错误的点上。通过这种scaling-up incorrect的操作，能够保证得到不同于g_t的g_{t+1}。

define scaling factor $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$

$$\begin{array}{lcll} \text{incorrect} & \leftarrow & \text{incorrect} & \cdot \ \blacklozenge_t \\ \text{correct} & \leftarrow & \text{correct} & / \ \blacklozenge_t \end{array}$$

- **equivalent** to optimal re-weighting
- $\blacklozenge_t \geq 1$ iff $\epsilon_t \leq \frac{1}{2}$
  —physical meaning: **scale up incorrect**; **scale down correct**
  —like what Teacher does

值得注意的是上述的结论是建立在$\epsilon_t \le \frac{1}{2}$的基础上，如果$\epsilon_t \ge \frac{1}{2}$，那么就做相反的推论即可。关于$\epsilon_t \ge \frac{1}{2}$的情况，我们稍后会进行说明。

从这个概念出发，我们可以得到一个初步的演算法。其核心步骤是每次迭代时，利用$\diamond t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$把$u_t$更新为$u_{t+1}$。具体迭代步骤如下：

$\mathbf{u}^{(1)} = ?$
for $t = 1, 2, \ldots, T$
1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where $\mathcal{A}$ tries to minimize $\mathbf{u}^{(t)}$-weighted 0/1 error
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, where ϵ_t = weighted error (incorrect) rate of g_t

return $G(\mathbf{x}) = ?$

但是，上述步骤还有两个问题没有解决，第一个问题是初始的$u^{(1)}$应为多少呢？一般来说，为了保证第一次E_{in}最小的话，设$u^{(1)} = \frac{1}{N}$即可。这样最开始的g_1就能由此推导。第二个问题，最终的G(x)应该怎么求？是将所有的g(t)合并uniform在一起吗？一般来说并不是这样直接uniform求解，因为g_{t+1}是通过g_t得来的，二者在E_{in}上的表现差别比较大。所以，一般是对所有的g(t)进行linear或者non-linear组合来得到G(t)。

- want g_1 'best' for E_{in}: $u_n^{(1)} = \frac{1}{N}$
- $G(\mathbf{x})$:
  - uniform? but g_2 very bad for E_{in} (why? :-))
  - linear, non-linear? as you wish

接下来的内容，我们将对上面的第二个问题进行探讨，研究一种算法，将所有的g(t)进行linear组合。方法是计算$g(t)$的同时，就能计算得到其线性组合系数α_t，即aggregate linearly on the fly。这种算法使最终求得g_{t+1}的时候，所有g_t的线性组合系数α也求得了，不用再重新计算α了。这种Linear Aggregation on the Fly算法流程为：

$\mathbf{u}^{(1)} = [\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}]$
for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where ...
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, where ...
3. compute $\alpha_t = \ln(\blacklozenge_t)$

return $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$

如何在每次迭代的时候计算α_t呢？我们知道α_t与ϵ_t是相关的：ϵ_t越小，对应的α_t应该越大，ϵ_t越大，对应的α_t应该越小。又因为$\diamond t$与ϵ_t是正相关的，所以，α_t应该是$\diamond t$的单调函数。我们构造α_t为：

$$\alpha_t = ln(\diamond t)$$

α_t这样取值是有物理意义的，例如当$\epsilon_t = \frac{1}{2}$时，error很大，跟掷骰子这样的随机过程没什么两样，此时对应的$\diamond t = 1$，$\alpha_t = 0$，即此g_t对G没有什么贡献，权重应该设为零。而当$\epsilon_t = 0$时，没有error，表示该g_t预测非常准，此时对应的$\diamond t = \infty$，$\alpha_t = \infty$，即此g_t对G贡献非常大，权重应该设为无穷大。

- wish: large α_t for 'good' g_t $\Longleftarrow$ $\alpha_t = \text{monotonic}(\blacklozenge_t)$
- will take $\alpha_t = \ln(\blacklozenge_t)$
  - $\epsilon_t = \frac{1}{2} \Longrightarrow \blacklozenge_t = 1 \Longrightarrow \alpha_t = 0$ (bad g_t zero weight)
  - $\epsilon_t = 0 \Longrightarrow \blacklozenge_t = \infty \Longrightarrow \alpha_t = \infty$ (super g_t superior weight)

这种算法被称为Adaptive Boosting。它由三部分构成：base learning algorithm A，re-weighting factor $\diamond t$和linear aggregation α_t。这三部分分别对应于我们在本节课开始介绍的例子中的Student，Teacher和Class。

Adaptive Boosting = weak base learning algorithm $\mathcal{A}$ (Student)
+ optimal re-weighting factor $\blacklozenge_t$ (Teacher)
+ 'magic' linear aggregation α_t (Class)

综上所述，完整的adaptive boosting（AdaBoost）Algorithm流程如下：

$\mathbf{u}^{(1)} = [\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}]$

for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where $\mathcal{A}$ tries to minimize $\mathbf{u}^{(t)}$-weighted 0/1 error
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by

   $[\![y_n \neq g_t(\mathbf{x}_n)]\!]$ (incorrect examples): $u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot \blacklozenge_t$

   $[\![y_n = g_t(\mathbf{x}_n)]\!]$ (correct examples): $u_n^{(t+1)} \leftarrow u_n^{(t)} / \blacklozenge_t$

   where $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$ and $\epsilon_t = \frac{\sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t)}}$
3. compute $\alpha_t = \ln(\blacklozenge_t)$

return $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$

从我们之前介绍过的VC bound角度来看，AdaBoost算法理论上满足：

- From VC bound

$$E_{\text{out}}(G) \le E_{\text{in}}(G) + O\left(\sqrt{\underbrace{O(d_{\text{vc}}(\mathcal{H}) \cdot T \log T)}_{d_{\text{vc}} \text{ of all possible } G} \cdot \frac{\log N}{N}}\right)$$

- **first term can be small**: $E_{\text{in}}(G) = 0$ after $T = O(\log N)$ iterations if $\epsilon_t \le \epsilon < \frac{1}{2}$ always
- **second term can be small**: overall d_{vc} grows "slowly" with T

上式中，$E_{out}(G)$的上界由两部分组成，一项是$E_{in}(G)$，另一项是模型复杂度O(*)。模型复杂度中$d_{vc}(H)$是g_t的VC Dimension，T是迭代次数，可以证明G的d_{vc}服从$O(d_{vc}(H) \cdot Tlog\,T)$。

对这个VC bound中的第一项$E_{in}(G)$来说，有一个很好的性质：如果满足$\epsilon_t \le \epsilon < \frac{1}{2}$，则经过$T = O(log\,N)$次迭代之后，$E_{in}(G)$能减小到等于零的程度。而当N很大的时候，其中第二项也能变得很小。因为这两项都能变得很小，那么整个$E_{out}(G)$就能被限定在一个有限的上界中。

其实，这种性质也正是AdaBoost算法的精髓所在。只要每次的$\epsilon_t \le \epsilon < \frac{1}{2}$，即所选择的矩g比乱猜的表现好一点点，那么经过每次迭代之后，矩g的表现都会比原来更好一些，逐渐变强，最终得到$E_{in} = 0$且E_{out}很小。

boosting view of AdaBoost:

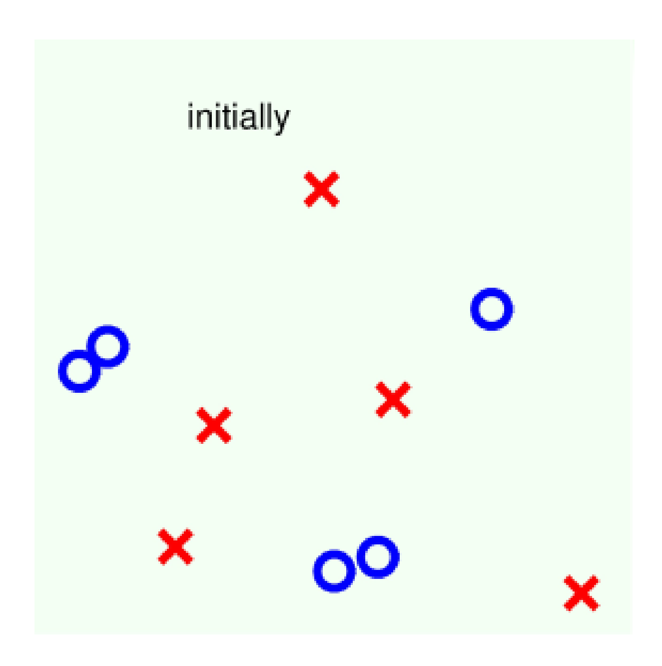
if $\mathcal{A}$ is weak but always **slightly better than random** ($\epsilon_t \leq \epsilon < \frac{1}{2}$),

then (AdaBoost+$\mathcal{A}$) can be strong ($E_{\text{in}} = 0$ and E_{out} small)

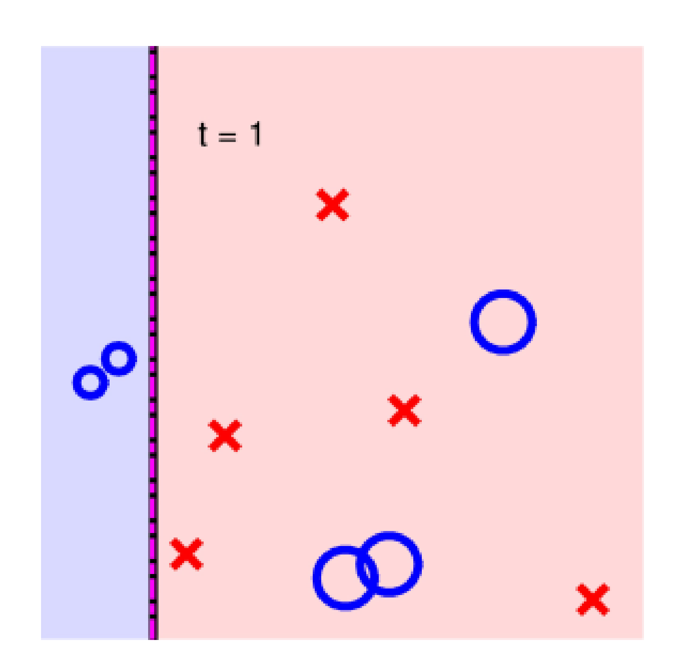
Adaptive Boosting in Action

上一小节我们已经介绍了选择一个“弱弱”的算法A（$\epsilon_t \leq \epsilon < \frac{1}{2}$，比乱猜好就行），就能经过多次迭代得到$E_{in} = 0$。我们称这种形式为decision stump模型。下面介绍一个例子，来看看AdaBoost是如何使用decision stump解决实际问题的。

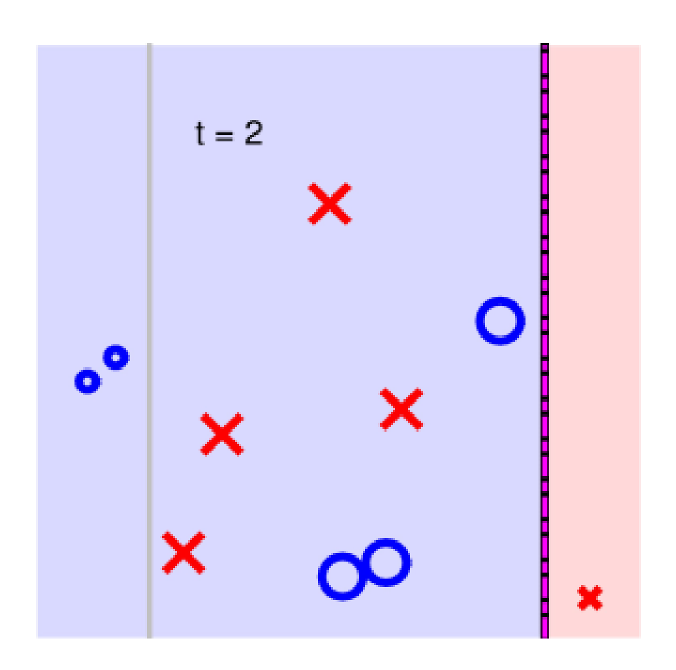
如下图所示，二维平面上分布一些正负样本点，利用decision stump来做切割。



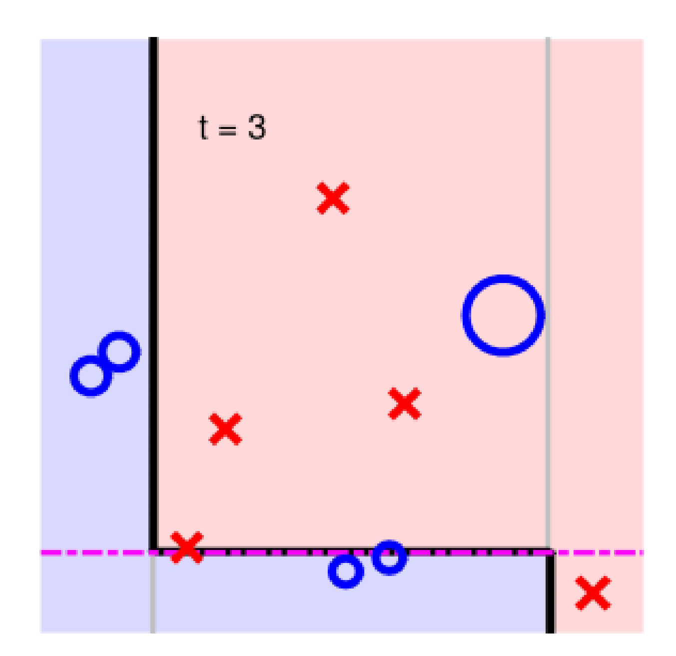
第一步：



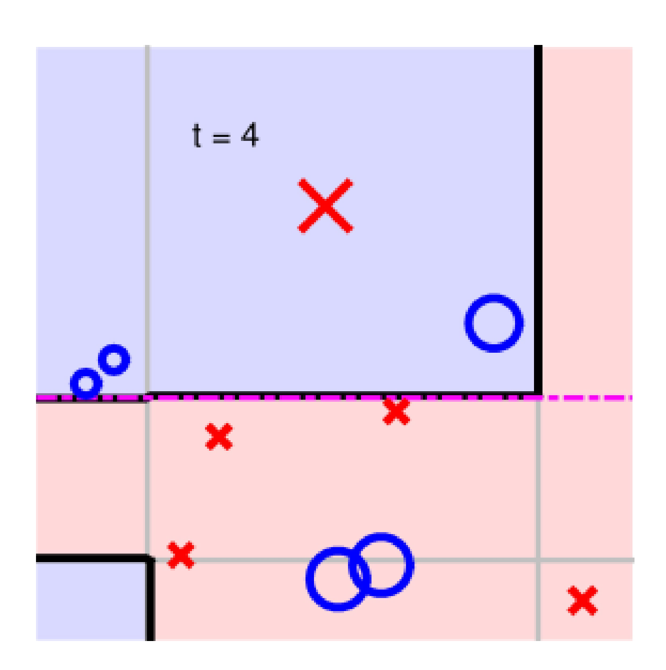
第二步:



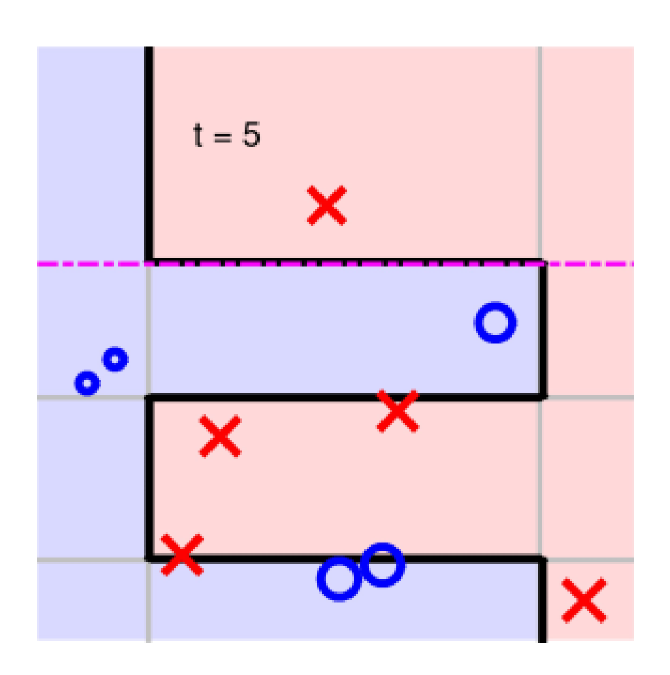
第三步:



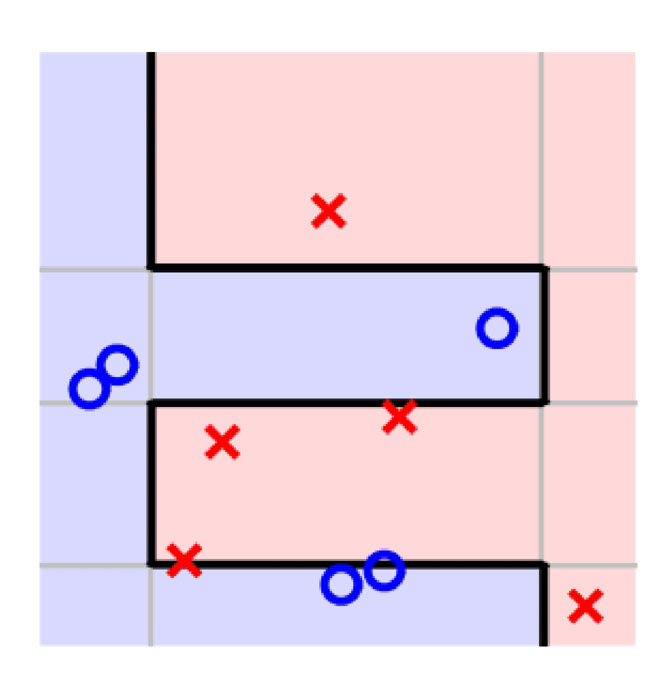
第四步:



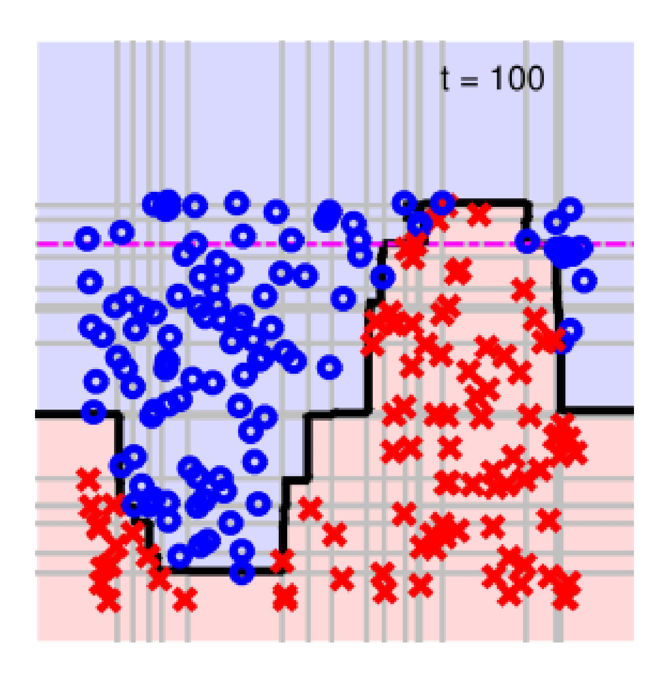
第五步:



可以看到，经过5次迭代之后，所有的正负点已经被完全分开了，则最终得到的分类线为：

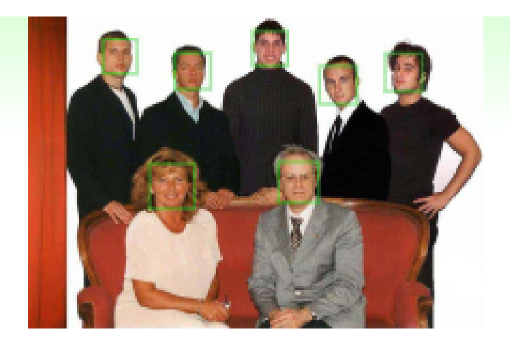


另外一个例子，对于一个相对比较复杂的数据集，如下图所示。它的分界线从视觉上看应该是一个sin波的形式。如果我们再使用AdaBoost算法，通过decision stump来做切割。在迭代切割100次后，得到的分界线如下所示。



可以看出，AdaBoost-Stump这种非线性模型得到的分界线对正负样本有较好的分离效果。

课程中还介绍了一个AdaBoost-Stump在人脸识别方面的应用：



original picture by F.U.S.I.A. assistant and derivative work by Sylenius via Wikimedia Commons

The World's First 'Real-Time' Face Detection Program

- AdaBoost-Stump as core model: linear aggregation of key patches selected out of 162,336 possibilities in 24x24 images
  —**feature selection** achieved through AdaBoost-Stump
- modified linear aggregation *G* to rule out non-face earlier
  —**efficiency** achieved through modified linear aggregation

总结

本节课主要介绍了Adaptive Boosting。首先通过讲一个老师教小学生识别苹果的例子，来引入Boosting的思想，即把许多“弱弱”的hypotheses合并起来，变成很强的预测模型。然后重点介绍这种算法如何实现，关键在于每次迭代时，给予样本不同的系数u，宗旨是放大错误样本，缩小正确样本，得到不同的小矩g。并且在每次迭代时根

据错误ϵ值的大小，给予不同g_t不同的权重。最终由不同的g_t进行组合得到整体的预测模型G。实际证明，Adaptive Boosting能够得到有效的预测模型。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程